Problem Set 3

ACA - Advanced C# 21/03/2024

Deadline

The deadline is **Wednesday, 27 March 2024 22:00:00**

Repository

You can find all necessary files and submission instructions in the following repository: Problem Set 3 Repo

Problem 1 - Dependency Injection

Background

Dependency Injection (DI) is a design pattern that allows for the creation of loosely coupled code by injecting dependencies at runtime rather than compile time. This pattern is especially useful in large-scale applications where managing dependencies manually becomes cumbersome and error-prone. A Dependency Injection Container (DIC) acts as a registry for all the dependencies in an application. It controls the creation and life cycle of dependent objects and their dependencies.

In this problem, you are tasked with implementing a basic Dependency Injection Container. The core functionality of your container will revolve around two interfaces: IServiceCollection and IServiceProvider. These interfaces will allow you to register services with their implementations and retrieve these services, respectively.

Problem Statement

Your task is to implement a basic Dependency Injection Container by creating classes that fulfill the contracts defined by the IAcaServiceCollection and IAcaServiceProvider interfaces. Your implementation should support two types of service lifetimes: Transient and Singleton.

Requirements

1. **Singleton Services:** For services registered with ServiceLifetime.Singleton, your container should create and return the same instance every time the service is requested.

2. **Transient Services:** For services registered with ServiceLifetime.Transient, your container should create a new instance each time the service is requested.

3. **Error Handling:** If a requested service is not registered in the container, return null. If any of its dependencies are not registered in the container, throw an exception.

Problem 2 - Extending the Dependency Injection Container

Background

After implementing the basic functionality of a Dependency Injection Container, it's time to enhance its usability and flexibility. One common approach to achieve this is through extension methods. Extension methods allow us to add new methods to existing types without creating a new derived type, recompiling, or otherwise modifying the original type. In the context of our Dependency Injection Container, extension methods can simplify the registration and retrieval of services.

Problem Statement

Your task is to extend the functionality of your Dependency Injection Container by writing extension methods for both IServiceCollection and IServiceProvider interfaces. These methods will leverage generic types to provide a more type-safe and concise way to register and retrieve services.

Implement the following extension methods for IServiceCollection:

1. **AddTransient<TService, TImplementation>:** Registers a transient service of the type specified in TService with an implementation type specified in TImplementation.

2. **AddTransient<TService>():** Registers a transient service of the type specified in TService with itself as the implementation.

3. **AddSingleton<TService, TImplementation>():** Registers a singleton service of the type specified in TService with an implementation type specified in TImplementation.

4. **AddSingleton<TService>():** Registers a singleton service of the type specified in TService with itself as the implementation.

Implement the following extension methods for IServiceProvider:

1. **TService? GetService<TService>():** Retrieves the service of the type specified in TService from the container. Returns null if the service is not registered.

2. **TService GetRequiredService<TService>():** Retrieves the service of the type specified in TService from the container. Throws an exception if the service is not registered.

These methods should utilize the GetService method defined in IServiceProvider, casting the result to TService.